AT91 ISP/SAM-BA®

User Guide

6421A–ATARM–04-Jun-09

Table of Contents

Section 1

Overview

1.1 Scope

The AT91 In-system Programmer (ISP) provides an open set of tools for programming Atmel® AT91SAM ARM® Thumb®-based microcontrollers. They are based on a common dynamic linked library (DLL), the AT91Boot_DLL. It is used by SAM-BA®, SAM-PROG and all ISP tools.

The SAM Boot Assistant (SAM-BA) software provides a means of easily programming different Atmel AT91SAM devices.

SAM-BA now uses AT91Boot_DLL.dll to communicate with the target.

This document describes how to extend SAM-BA capabilities to program any kind of memory and provides a helpful guide to installing and using the AT91 ISP.

1.2 Key Features of the SAM-BA Software

- Performs in-system programming through JTAG, RS232 or USB interfaces
- Provides both AT91SAM embedded flash programing and external flash programing solutions
- May be used via a Graphical User Interface (GUI) or started in batch mode from a DOS window
- Runs under Windows® 2000 and XP
- Memory and peripheral display content
- User scripts executable from SAM-BA Graphical User Interface or a shell

Section 2

Running SAM-BA

2.1 Overview

SAM-BA can operate in a graphical mode or it can be launched in command line mode with a TCL script in parameter. Both modes can be combined to easily obtain a powerful loading solution on AT91SAM devices customized for the current project.

2.2 Running SAM-BA

Connect your board to your communication interface (either the host serial COM port, or the USB device port, or the SAM-ICE™ JTAG probe).

Warning: The USB cable must NOT be connected to the board for an RS232 use, otherwise the USB interface is chosen by default.

There are two different ways to start SAM-BA:

1. Click on the SAM-BA icon

or

2. Type in a shell:

> [*Install Directory*]/SAM-BA.exe [Communication Interface] [Board] [Script_File] [Script_File Args]

where:

- [*Communication Interface*]: “\usb\ARM0” for USB, “\jlink\ARM0” for JTAG, ”COMx” for RS232 where x is the COM port number
- [*Board*]: the name of the board accessible through the Choose Protocol window (see Figure 2-1)
- *[Script_File*] (Optional): Path to the TCL Script File to execute
- [*Script_File Args*] (Optional): TCL Script File Arguments

Depending on the number of arguments, SAM-BA opens in different modes. If [Communication Interface] and [Board] arguments are provided in the command line, SAM-BA GUI will start directly without displaying the connection message box. If [*Script_File*] argument is not specified, SAM-BA starts in GUI mode.

Some valid command line examples are listed below:

- launch SAM-BA with script file and one argument:

```
SAM-BA.exe \usb\ARM0 AT91SAM9261-EK lib/historyCommand.tcl AT91C_DBGU_THR
```

- launch SAM-BA with script file:

```
SAM-BA.exe \usb\ARM0 AT91SAM9261-EK ./myScript.tcl
```

- launch SAM-BA GUI with preselected communication link and board:

```
SAM-BA.exe \usb\ARM0 AT91SAM9261-EK
```

It is possible to catch information sent to standard outputs by SAM-BA while executing in command line mode. This can be achieved with the following command:

```
SAM-BA.exe \usb\ARM0 AT91SAM9261-EK ./myScript.tcl > log.txt
```

2.3 SAM-BA GUI

When SAM-BA is launched, after selection of the board and the communication link, the main window appears (see Figure 2-2). It contains three different areas. From top to bottom, they are:

- Memory Display area
- Memory Download area
- TCL Shell area

The Memory Display and the Memory Download areas are used to simplify the memory access.

2.3.1 Board Connection

SAM-BA scans active USB connections with AT91SAM based boards, it is mandatory to connect the target to the PC before launching SAM-BA. Non USB connections do not benefit from autodetection.

Warning: The USB cable must NOT be connected to the board for RS232 use, otherwise the USB interface is chosen by default.

Note: To change the connection type (RS232/USB/JTAG), the target must be rebooted and SAM-BA must be restarted.

When SAM-BA starts, a pop-up window (see Figure 2-1) appears that enables selection of the board. Likewise, the connection has to be selected from the dropdown menu:

"\usb\ARMX" for USB connected devices

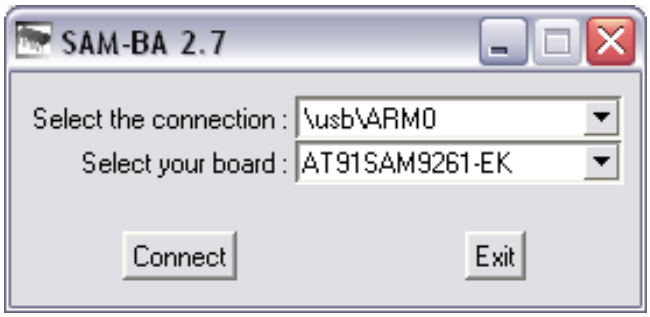
"\jlink\ARMX" for SAM-ICE/JLink connected devices

"COMX" for available COM ports

Note: To select another board, SAM-BA must be restarted.
If no USB connected device is available in the list, please check that devices have been connected before SAM-BA was launched.

Warning: JTAG connection with a new board is possible as soon as the oscillator is the same as the corresponding AT91SAM-EK one.

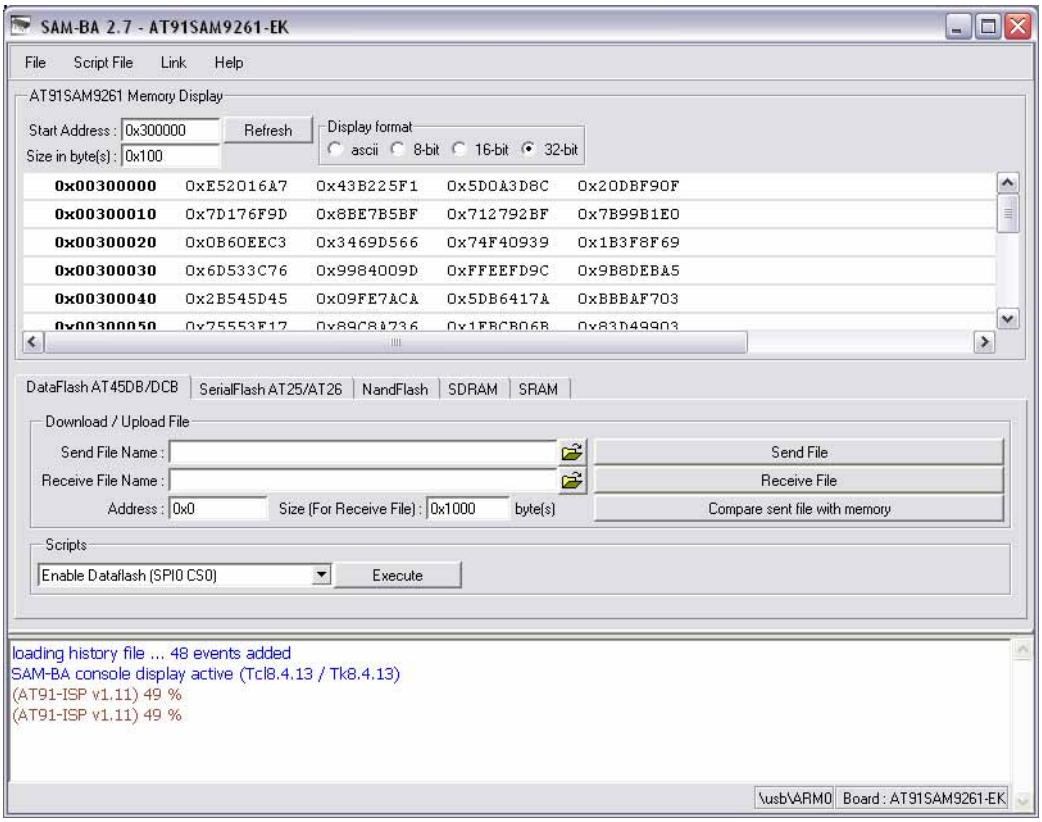
Figure 2-1. Choose Protocol Window



Once SAM-BA's main window is displayed, the name of the board is shown on the right side of the status bar (see Figure 2-2).

Click on the Connect button. The main window appears (see Figure 2-2).

Figure 2-2. SAM-BA Main Window

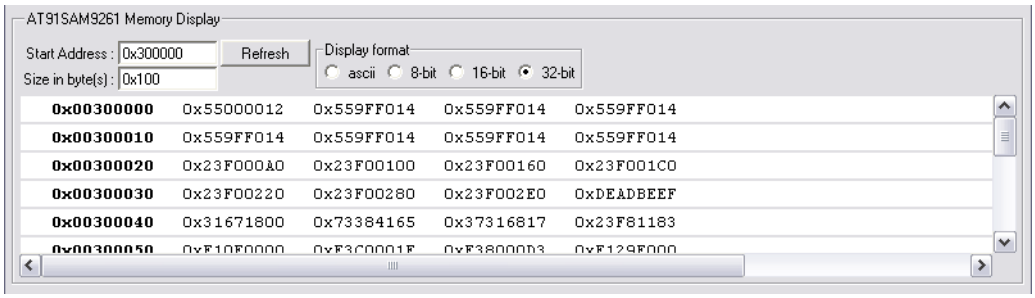


2.3.2 Memory Display Area

In this area you can display a part of the microcontroller memory content. Three different display formats can be used: 32-bit word, 16-bit half-word or 8-bit byte, with a maximum display of 1024-byte long memory area. Values can also be edited by double-clicking on them (see Section 2.3.2.2 "Edit Memory Content").

Note: Only valid memory areas or system/user peripheral areas are displayed. An error message is written in the TCL Shell area (see, Section 2.3.4 "TCL Shell Area") if a forbidden address is supplied or if a memory overrun occurs.

Figure 2-3. Memory Display Area



2.3.2.1 Read Memory Content

1. Enter the address of the area to read in the Starting Address field.

   Note: If a wrong address value is entered, an error message is displayed in the TCL Shell area.

2. Enter the size of the area to display.

   Note: If a wrong size is entered, an error message is displayed in the TCL Shell area.

3. Choose a display format: 32-bit word, 16-bit half-word or 8-bit byte. This automatically refreshes the memory contents.
4. Press the Refresh button.

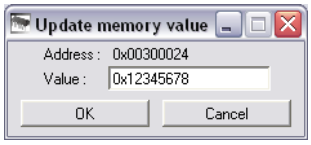
2.3.2.2 Edit Memory Content

Some memories and/or embedded peripherals can be edited:

1. Double-click on the value to update it. An editable pop-up window appears (see Figure 2-4).

   Note: Only memories can be updated this way, e.g., static RAM or SDRAM (if previously initialized). If you try to write the other memory types, nothing happens.

Figure 2-4. Update Memory Value Window



2. Press OK to update the value in the ***Memory display*** area. The corresponding TCL command is displayed in the TCL Shell area.

   Note: Only the lowest bits of the value are taken into account if the format of the value entered is higher than the display format.

2.3.3 Memory Download Area

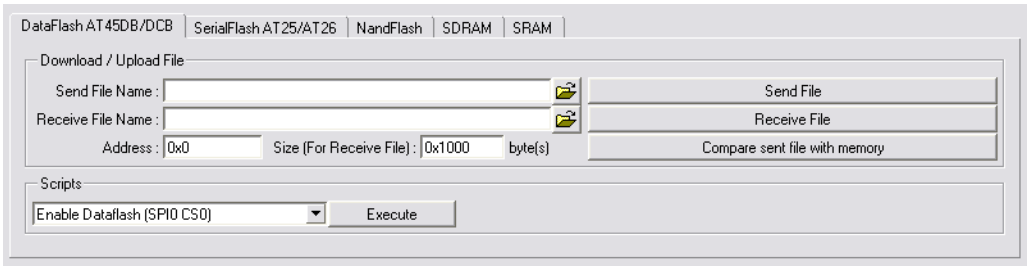
The ***Memory download*** area provides a simple way to upload and download data into internal and external memories.

For each memory, files can be sent and received, and the target's memory content can be compared with a file on your computer (see Figure 2-5).

Note: Only binary file format is supported by SAM-BA.

This area also gives access to some specific scripts for the different memories available on the board (Nand Flash, DataFlash®, etc.).

Figure 2-5. Memory Download Area



2.3.3.1 Initialize Memory

To be able to program data into the internal Flash memory or into the external memory, SAM-BA uses a small program called: applet. This applet is specific to each memory and device.

2.3.3.2 Upload a File

- First, select the memory by clicking on its corresponding tab.
- Enter the file name location in the Send File name field or open the file browser by clicking on the Open Folder button and select it.

  If you enter a wrong file name, an error message will be displayed in the TCL Shell (see Section 2.3.4 ”TCL Shell Area”).
- Enter the destination address in the selected memory where the file should be written.

  If you enter a forbidden address, or if your file overruns the memory size, an error message is displayed in the TCL Shell.

  Note: A forbidden address corresponds to an address outside the selected memory range address.
- Send the file using the *Send File* button. Make sure that the memory is correctly initialized before sending any data.

2.3.3.3 Download Data to a File

- First, select the memory by clicking on its corresponding tab.
- Enter the file name location in the Receive File name field or open the file browser by clicking on the Open Folder button and select it.

  If you enter a wrong file name, an error message is displayed in the TCL Shell (see Section 3.7).
- Enter the address of the first data to read in the Address field.
- Enter the data size to read in the Size field.

  If you enter a forbidden address, or if your file size overruns the memory size, an error message is displayed in the TCL Shell.

  Note: A forbidden address corresponds to an address outside the selected memory range address.

Get data using the *Receive File* button. Make sure that your memory is correctly initialized before getting any data.

2.3.3.4 Compare Memory with a File

Usually, this feature allows to check if a sent file was correctly written into the memory, but you can compare any file with your memory content. The comparison is made on the size of the selected file.

- First, select the concerned memory by clicking on its corresponding tab.
- Enter the file name location in the *Send File* name field or open the file browser by clicking on the *Open Folder* button and select it.

  If you enter a wrong file name, an error message will be displayed in the TCL Shell (see Section 2.3.4 ”TCL Shell Area”).
- Enter the address of the first data to compare with the selected file in the Address field.

  If you enter a forbidden address, or if your file size overruns the memory size, an error message is displayed in the TCL Shell.

Note: A forbidden address corresponds to an address outside the selected memory range address.

- Compare the selected file with the memory content using the *Compare sent file with memory* button. A message box is displayed if the file matches or not with the memory content of the file size. Make sure that your memory is correctly initialized before comparing any data.

Figure 2-6. Comparison Result Successful

Figure 2-7. Comparison Result Failed

2.3.3.5 Memory Scripts

Some scripts may be supplied for each memory. Usually, these scripts allow you to configure and initialize quickly the corresponding memories (SDRAM initialization, erase all Flash, etc).

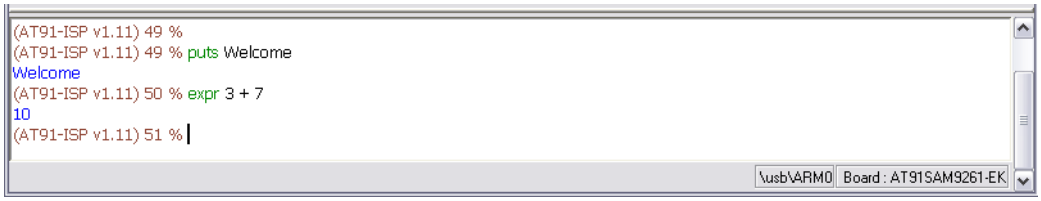
- To execute a script, select the memory by clicking on its tab.
- Select the script to launch in the list box.
- Click on the Execute button.

Note: Messages which inform of the correct execution of the script are displayed in the TCL Shell (Section 2.3.4 ”TCL Shell Area”) and/or through a message box.

2.3.4 TCL Shell Area

This is a standard TCL shell. Everything you type in the shell is interpreted by a TCL interpreter. This area gives you access to standard TCL commands. Type “*puts Welcome*” and you will get the result “*Welcome*”, type “*expr 3 + 7*” and you will get the result “*10*” (see Figure 2-8).

Figure 2-8. TCL Shell Window



TCL is a commonly used scripting language for automation. This interpreted language offers a standard set of commands which can be extended by application specific commands written in C or other languages. Tutorials and a manual can be downloaded here: http://www.tcl.tk/doc/.

Specific commands have been added to the SAM-BA TCL interpreter to interface with AT91SAM devices. These basic commands can be used to easily build more complex routines.

SAM-BA allows you to create, edit and execute script files. A script file configures your device easily or automatically runs significant scripts.

The *Script File* menu supplies commands to start and stop recording, to execute, reset, edit and save the recording file.

The name of the generated file is *historyCommand.tcl*.

2.3.4.1 Start/Stop/Reset Recording

In the *Script File* menu, select *Start Recording* to begin the record. Now, all the commands that are to be executed in the different blocks of the software are recorded in a specific file called ***history Command.tcl***. This file is located in the user directory.

Note: Only this file can be written through the Start/Stop/Reset Recording commands.

New recorded commands are added at the end of the ***historyCommand.tcl*** file.

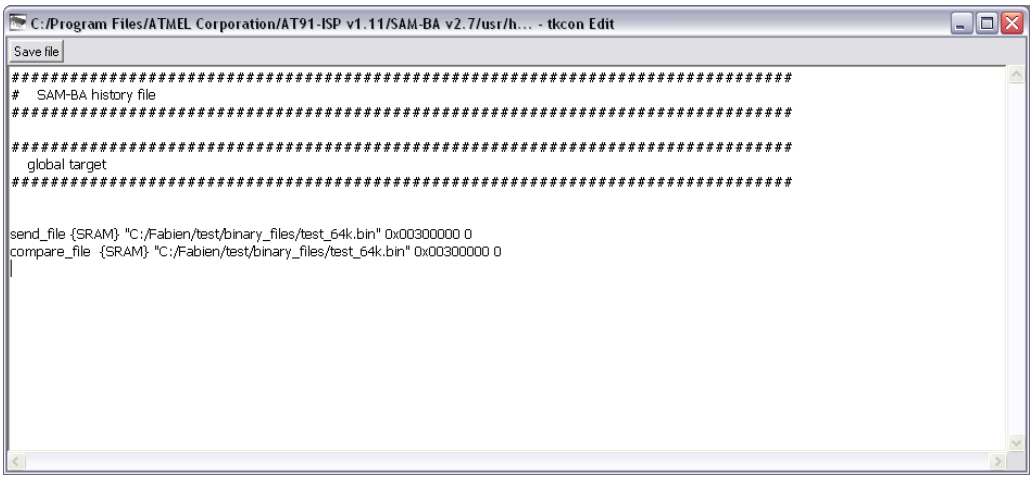
To stop recording, select Stop Recording in the menu.

To erase the ***historyCommand.tcl*** content, select Reset Recording.

2.3.4.2 Editing Script Files

SAM-BA embeds ***historyCommand.tcl*** recording file editor. *Edit Script File* command in the *Script File* menu launches the editor. A new window appears to edit and save the history contents. Modified script can be saved using another file name through the Save file button.

Figure 2-9. Script File Edition View



2.3.4.3 Script File Execution

There are two possibilities to execute a script file.

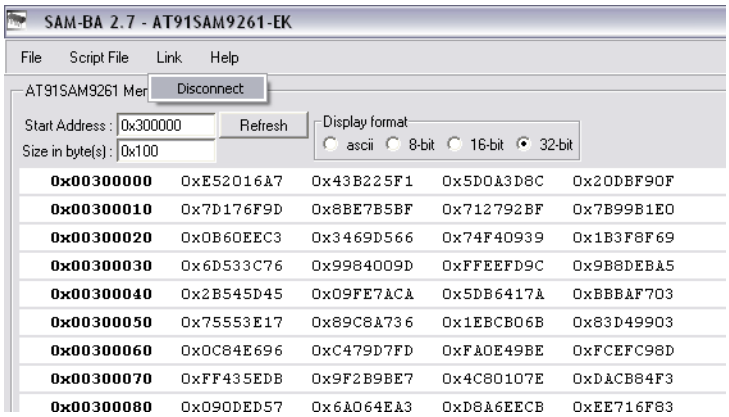
- See Section 2.4 "Running SAM-BA Using the Command Line" for more information on how to execute TCL script files from a shell.
- Use the command *Execute Script File* in the GUI *Script File* menu and enter the TCL file to execute. Messages that inform of the correct execution of the script are displayed in the TCL Shell and/or through message boxes.

Note: All TCL commands can be executed through script files.

2.3.5 Disconnect

It is possible to close the current connection if necessary by clicking on the corresponding *Link/Disconnect* menu (see Figure 2-10).

Figure 2-10. Link Menu



2.4 Running SAM-BA Using the Command Line

Most standard editors offer shortcuts to run DOS command lines. It is possible to run SAM-BA using the following command line:

[Install Directory]/SAM-BA.exe [*Communication Interface*] [*Board*] [*Script.tcl*] [*args]* [> logfile.log]

where:

- [*Communication Interface*]: \usb\ARM0, COMx (for RS232) where x is the COM port number, or \jlink\ARM0
- [*Board*]: the name of the board accessible through the Choose Protocol window (COM1, COM2, USB, etc.)
- [*Script.tcl*]: the name of a TCL script file to be immediately executed
- [*args*]: Arguments to be transferred to the TCL script
- *[> logfile.log*]: optional argument to get the log of the script execution in a file.

For example: `> C:\Sam-ba.exe COM2 AT91SAM7S64-EK load_app.tcl > result.log`

If bad arguments are entered in the command line or if there are communication problems, SAM-BA is not able to start.

If the *Script.tcl* argument is not specified, then the SAM-BA GUI will automatically appear. Otherwise, the TCL script is executed in the same TCL interpreter environment as is run in the TCL shell area of the SAM-BA GUI.

2.5 Creating a Script

2.5.1 Rapid Script Generation

A very simple way to record a script file is to launch SAM-BA in GUI mode, and use the ScriptFile menu. Start the recording on script, and then, every action done in the GUI (changing a value in the Memory Display area, executing a script in a Memory Download window, or sending/receiving file) is recorded in the historyCommand.tcl file. This file can be used as is when launching SAM-BA in command line mode. It can also be used as a starting point to elaborate longer scripts.

This mode is very useful for automating memory programming.

2.5.2 SAM-BA Open Scripting Environment

When SAM-BA is launched in any mode, a board must be specified. This information is required to set up the scripting environment (can be retrieved by reading target(board) global variable).

A set of variable definitions and a set of memory algorithms correspond to each board. Variable definitions correspond to the standard LibV3 symbols. Using symbols instead of absolute values increases the readability and re-use of the TCL routines.

Note: These variables are declared as global variables. Within functions, symbols must be declared as global to reference the global variables:

```
proc foo {
  global target
  set err_code 0
  global AT91C_DBGU_THR

  TCL_Write_Int $target(handle) 64 $AT91C_DBGU_THR err_code
}
```

In the same way, TCL programing routines can be invoked for loading code into the device.

For a better understanding of the TCL libraries, all programing functions in TCL are delivered in the SAM-BA installation directory:

```
[Install Directory]\SAM-BA v2.x\lib\common
```

2.5.2.1 SAM-BA Scripts for Demos

Most AT91 demos (Linux®, Windows CE®) are delivered with a TCL script to deploy binaries in AT91SAM devices memories. These scripts can be used as a startpoint to deploy custom application on Atmel AT91SAM Evaluation Kits or AT91SAM-based boards.

Section 3

AT91 ISP Architecture

3.1 Overview

The AT91 In-system Programmer (ISP) provides an open set of tools for programming the AT91SAM7, AT91SAM9, and ATSAM3 ARM-based microcontrollers. They are based on a common dynamic linked library (DLL), the AT91Boot_DLL. It is used by SAM-BA, SAM-PROG™ and all ISP tools.

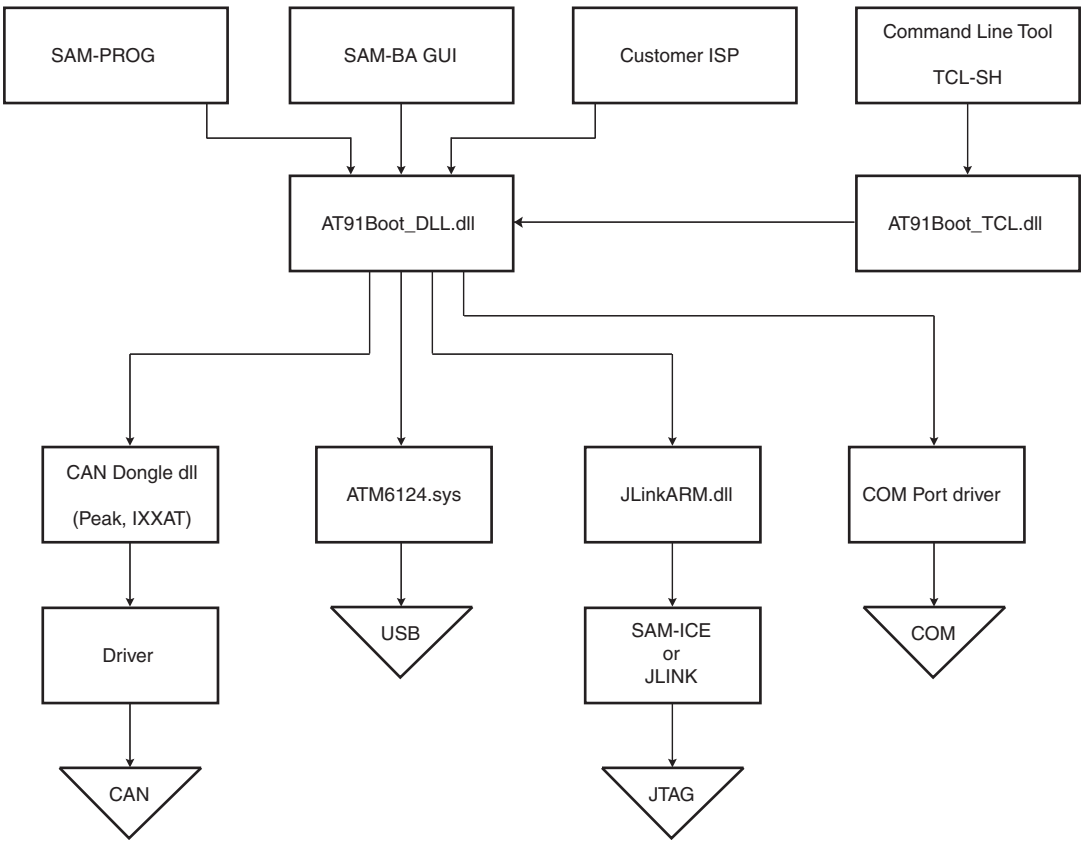
AT91Boot_DLL API is in the public domain so that custom GUI ISP solutions can be built. It avoids writing low-level functions such as Flash memory writing algorithms, etc.

AT91Boot_DLL is an OLE COM component distributed under a DLL (AT91Boot_DLL.dll) allowing automation tools.

It is also possible to execute the AT91Boot_DLL functions in command lines in a TCL shell. An intermediate DLL (AT91Boot_TCL.dll) is used to transform TCL commands into calls to AT91Boot_DLL.

Several communication links are available such as USB, serial link, CAN or JTAG.

Figure 3-1. AT91 ISP Framework Architecture



3.2 DLL Prerequisites

- Runs under Windows 2000/XP
- A SAM-ICE or a JLink JTAG box and its associated USB drivers (only necessary to use JTAG communication link)
- CAN Dongles
  - PCAN-USB Peak dongle
  - USB-to-CAN compact IXXAT dongle
- TCL Toolchain including tclsh can be downloaded from the following URL: http://www.activestate.com/Products/ActiveTcl/

3.3 AT91 ISP Installation

Installation is automatic using the AT91_ISP vx.yy.exe install program.

3.4 Contents

3.4.0.1 Library Directory

All files located in the Library directory are necessary for the AT91Boot_DLL to run correctly.

- AT91Boot_DLL.dll
- AT91Boot_DLL.tlb type library file
- JLinkARM.dll
- AT91Boot_TCL.dll
- CAN Dongle dlls

3.4.0.2 Examples Directory

This directory contains some example projects using AT91Boot_DLL.dll. See the section <Blue>“Using AT91Boot_DLL Project Examples” for more information on the following projects:

- OLE_MFC project under Visual C++ 6.0
- OLE_without_MFC project under Visual C++ 6.0
- CAN_TCLSH gives an example of a TCL script that can be used to program a SAM7X256-based board over the CAN network.

3.4.0.3 SAM-PROG Application

This application downloads a binary file into the Flash memory of one or more AT91SAM devices in parallel from a PC or JTAG probe.

3.4.0.4 SAM-BA Boot4CAN Directory

This directory contains binary files for AT91SAM7A3 and AT91SAM7X devices. These files must be programmed into internal Flash memory before communicating over a CAN. SAM-PROG can be used to program these files.

3.4.1 DLL Registration

AT91Boot_DLL needs to be registered in the Windows Base Register in order to be used correctly. The Install program will register AT91Boot_DLL automatically.

AT91Boot_DLL.dll uses JLinkARM dll. In order for the user to compile a project anywhere, “YOUR_INSTALL_DIRECTORY\Library” path has been added to the PATH user environment variable. If it is not the case, JLinkARM dlls has to be set in the current directory of your application in order to be found by the AT91Boot_DLL.

Note: It is also possible to copy dll contained in the Library directory into WINNT/System32 as this directory is in the PATH environment variable by default. Do not forget to register AT91Boot_DLL after moving.

To register AT91Boot_DLL manually, execute the following command from a DOS Window or directly through the Windows Start/Execute menu:

```
regsvr32 /s /c "YOUR_INSTALL_DIRECTORY\AT91Boot_DLL.dll"
```

Note: regsvr32.exe is located in WINNT/System32 directory

3.4.2 Updating JLink/SAM-ICE Software

In order to function correctly, compatibility between JLink/SAM-ICE firmware, USB drivers and JLinkARM DLL is necessary. Thus it is recommended to update JLink/SAM-ICE software.

The JLink/SAM-ICE software update, contained in a zip file, is available on the www.segger.com web site in the “Downloads”, then “J-Link ARM” sub-areas. To proceed with update, carry out the following steps:

- Download the “Jlink_ARM” zip file.
- Unzip this download.
- Run the .exe file contained in it.
- Check the update in the “Doc\ReleaseNotes”.
- Run the new J-Link.exe to update the JLink/SAM-ICE firmware.
- Check if your PC driver is up to date with the delivery driver in the “USBDriver” folder contained in the .exe.
- Copy the JLinkARM.dll DLL to “YOUR_INSTALL_DIRECTORY\Library\” folder.

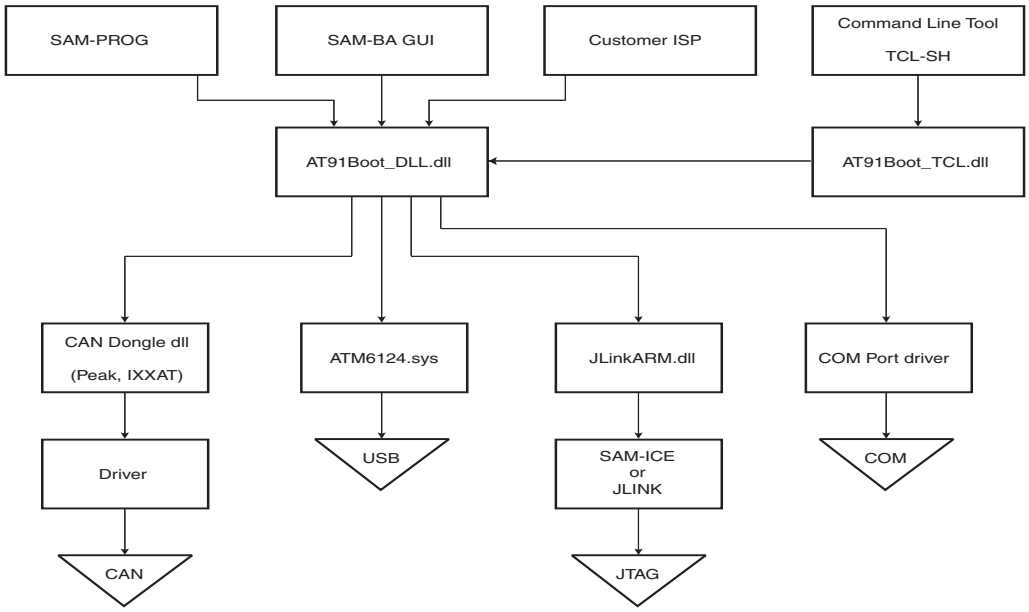
This completes the software update.

3.5 Communicating with AT91SAM Devices

3.5.1 Communication Links

AT91Boot_DLL connects AT91SAM-based targets through a USB link, a serial link or a JTAG using a SAM-ICE or JLink JTAG box.

Figure 3-2. Different Ways of Communicating with AT91SAM-based Targets



Depending on which communication link is selected, the target must be in the following state:

- When using the USB link or the DBGU serial link, SAM-BA Boot must run onto the target.
- When using the CAN link, SAM-BA Boot4CAN must run onto the target.
- When using JTAG communication through SAM-ICE or JLink, the target may be in an undefined state. In this case, it is up to the user to configure the target (PLL, etc.) if necessary.

3.5.2 Starting Communication

The AT91Boot_DLL principle is simple. It consists of:

1. Scanning all devices connected to the PC
2. Opening communication to the selected device
3. Performing all desired actions such as writing into Flash memory
4. Closing communication

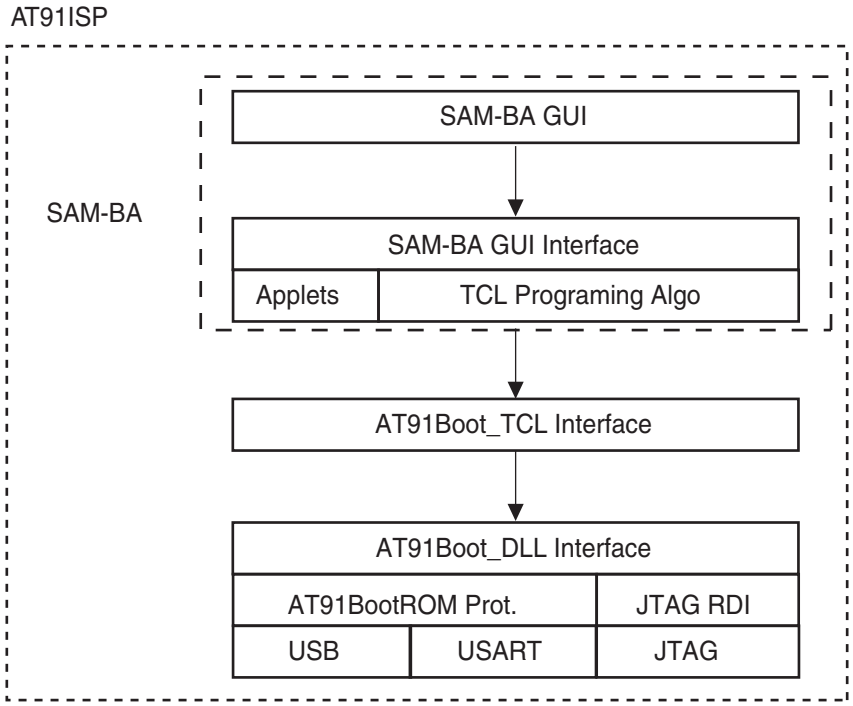
3.6 TCL Scripting Language

TCL is a commonly used scripting language for automation. This interpreted language offers a standard set of commands which can be extended with application specific commands written in C or other languages. Tutorials and manuals can be downloaded here: http://www.tcl.tk/doc/.

Specific commands have been added to the SAM-BA TCL interpreter to interface with AT91SAM devices. These basic commands can be used to easily build more complex routines.

In order to communicate with the board, API functions are available to deal with AT91Boot_DLL.dll library.

Figure 3-3. AT91ISP Levels



3.7 AT91Boot_DLL Interface

3.7.1 Low-level Functions

A description and a code example is given for each function.

These functions are available for all AT91SAM microcontrollers.

3.7.1.1 AT91Boot_Scan

This function scans connected devices and returns a list of connected devices. Detection is performed in the following order:

1. USB connected devices using ATM6124.sys driver
2. Connected SAM-ICE or JLink devices
3. CAN dongles (Peak, IXXAT)
4. All available serial COM ports

Note: The AT91Boot_Scan function does not verify if an Atmel device is really present, so even if there are no Atmel devices connected to SAM-ICE/JLink devices, CAN dongles or COM ports, these connections are returned in the connected devices list. This does not concern USB devices.

3.7.1.1.1 Description

*void AT91Boot_Scan(char *pDevList)*;

Table 3-1. AT91Boot_Scan

Type	Name	Details
Input Parameters	char *pDevList	Pointer to a char* table. All table entries must have been allocated prior using the AT91Boot_Scan function. (1)
Output Parameters	char *pDevList	Strings returned in the table: - *"\usb\ARMX"* for USB connected devices - *"\jlink\ARMX"* for SAM-ICE/JLink connected devices - *"\can\AtCanPeak\ARM"* for PCAN-USB Peak connected dongle - *"\can\Ixxat\ARM"* for USB-to-CAN compact IXXAT connected dongle - *"COMX"* for available COM ports
Return Code	none	

Note: 1. Each string must be allocated from the application and must have a size superior to 80 bytes. That string is used to recover, in particular CAN dongle, USB or JTAG box device name which is then replaced by a reduced symbolic name.

3.7.1.1.2 Code Example

```
CHAR *strConnectedDevices[5];
for (UINT i=0; i<5; i++)
  strConnectedDevices[i] = (CHAR *)malloc(100);
AT91Boot_Scan((char *)strConnectedDevices);
```

AT91Boot_Scan may return code similar to that below:

```
strConnectedDevices[0] : \usb\ARM0
strConnectedDevices[1] : \usb\ARM1
strConnectedDevices[2] : \jlink\ARM0
strConnectedDevices[3] : \can\AtCanPeak\ARM
strConnectedDevices[4] : COM1
```

3.7.1.2 AT91Boot_Open

This function opens the communication link on an AT91SAM device depending on the string given in the argument:

- USB
- JTAG
- CAN
- Serial COM port

Note: At this step, the Atmel device MUST be connected to either SAM-ICE/JLink, CAN network or COM port if using such a communication link.

3.7.1.2.1 Description

*void AT91Boot_Open(char *name, int *h_handle)*;

Table 3-2. AT91Boot_Open

Type	Name	Details
Input Parameters	*name	Pointer to a string returned by AT91Boot_Scan function[1]
Output Parameters	*h_handle	Communication handle: - NULL if opening connection failed - Non NULL if opening connection succeeded
Return Code	void	

Note: 1. As AT91Boot_Scan function detects only CAN dongles and not AT91SAM devices which are connected to, it is recommended to add an identifier to the end of string for each device such as, for example, *“\can\AtCanPeak\ARM0“*, *“\can\AtCanPeak\ARM1“...*

3.7.1.2.2 Code Example

```
AT91Boot_Open(strConnectedDevices[0], &h_handle);
AT91Boot_Open(''\can\AtCanPeak\ARM0'', &h_handle);
```

3.7.1.3 JTAG Communication Link

When opening a JTAG communication link through a SAM-ICE or a JLink by using the following command:

```
AT91Boot_Open(''\jlink\ARM0'', &h_handle);
```

the following steps are performed:

1. Open JLinkARM.dll and its associated library functions.
2. Set JTAG speed to 5 kHz in order to connect to the target even if it is running at 32 kHz.
3. Stop the target.
4. Set a hardware breakpoint at address 0.
5. Send a PROCRST command (RSTC_CR) in the Reset Controller in order to disable the Watchdog.
6. Wait for the target to reach the breakpoint.
7. Download a monitor into the target internal SRAM that allows communication only through the ARM Debug Communication Channels(1) by using the SAM-BA Boot commands(2).
8. Jump to the monitor in internal SRAM. Then monitor switches on the Main Oscillator(3).
9. For AT91SAM7, theJTAG speed is set to 3 MHz as it is the lowest allowed crystal frequency. For AT91SAM9, the JTAG clock is in adaptive mode.

Note:
1. For further information about DCC, visit www.arm.com.
2. For further information about SAM-BA Boot commands, see the Boot Program section of the product datasheet.
3. It is recommended to configure the PLL when returning from AT91Boot_Open function in order to speed up monitor execution.

3.7.1.4 AT91Boot_Close

This function closes the communication link previously opened on an AT91SAM device.

3.7.1.4.1 Description

void AT91Boot_Close(int h_handle);

Table 3-3. AT91Boot_Close

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
Output Parameters	none	
Return Code	void	

3.7.1.4.2 Code Example

```
AT91Boot_Close(h_handle);
```

3.7.1.5 AT91Boot_Write_Int

This function writes a 32-bit word into the volatile memory of the connected target.

3.7.1.5.1 Description

*void AT91Boot_Write_Int(int h_handle, int uValue, int uAddress, int *err_code);*

Table 3-4. AT91Boot_Write_Int

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	uValue	32-bit value to write
	uAddress	Address where to write 32-bit value
Output Parameters	none	
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.5.2 Code Example

```
AT91Boot_Write_Int(h_handle, 0xCAFECAFE, 0x200000, &err_code);
```

3.7.1.6 AT91Boot_Write_Short

This function writes a 16-bit word into the volatile memory of the connected target.

3.7.1.6.1 Description

*void AT91Boot_Write_Short(int h_handle, short wValue, int uAddress, int *err_code);*

Table 3-5. AT91Boot_Write_Short

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	wValue	16-bit value to write
	uAddress	Address where to write 16-bit value
Output Parameters	none	
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.6.2 Code Example

```
AT91Boot_Write_Short(h_handle, 0xCAFE, 0x200000, &err_code);
```

3.7.1.7 AT91Boot_Write_Byte

This function writes an 8-bit word into the volatile memory of the connected target.

3.7.1.7.1 Description

*void AT91Boot_Write_Byte(int h_handle, char bValue, int uAddress, int *err_code);*

Table 3-6. AT91Boot_Write_Byte

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	bValue	8-bit value to write
	uAddress	Address where to write 8-bit value
Output Parameters	none	
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.7.2 Code Example

```
AT91Boot_Write_Byte(h_handle, 0xFE, 0x200000, &err_code);
```

3.7.1.8 AT91Boot_Write_Data

This function writes X bytes into the volatile memory of the connected target.

3.7.1.8.1 Description

*void AT91Boot_Write_Data(int h_handle, int uAddress, char *bValue, int uSize, int *err_code);*

Table 3-7. AT91Boot_Write_Data

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	uAddress	Address where to write 8-bit value
	*bValue	Pointer to 8-bit data buffer to write
	uSize	Buffer size in bytes
Output Parameters	none	
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF004): USART Communication link not opened • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.8.2 Code Example

```
char bData[10] =
{0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09};

AT91Boot_Write_Data(h_handle, 0x200000, bData, 10, &err_code);
```

3.7.1.9 AT91Boot_Read_Int

This function reads a 32-bit word from the connected target.

3.7.1.9.1 Description

*void AT91Boot_Read_Int(int h_handle, int *uValue, int uAddress, int *err_code);*

Table 3-8. AT91Boot_Read_Int

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	*uValue	Pointer to a 32-bit value
	uAddress	Address where to read 32-bit value
Output Parameters	*uValue	32-bit read value
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.9.2 Code Example

```
int ChipId;
AT91Boot_Read_Int(h_handle, &ChipId, 0xFFFFF240, &err_code);
```

3.7.1.10 AT91Boot_Read_Short

This function reads a 16-bit word from the connected target.

3.7.1.10.1 Description

*void AT91Boot_Read_Short(int h_handle, short *wValue, int uAddress, int *err_code);*

Table 3-9. AT91Boot_Read_Short

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	*wValue	Pointer to a 16-bit value
	uAddress	Address where to read 16-bit value
Output Parameters	*wValue	16-bit read value
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.10.2 Code Example

```
short wRead;
AT91Boot_Read_Short(h_handle, &wRead, 0x200000, &err_code);
```

3.7.1.11 AT91Boot_Read_Byte

This function reads an 8-bit word from the connected target.

3.7.1.11.1 Description

*void AT91Boot_Read_Byte(int h_handle, char *bValue, int uAddress, int *err_code);*

Table 3-10. AT91Boot_Read_Byte

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	*bValue	Pointer to an 8-bit value
	uAddress	Address where to read 16-bit value
Output Parameters	*bValue	8-bit read value
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.11.2 Code Example

```
char bRead;
AT91Boot_Read_Byte(h_handle, &bRead, 0x200000, &err_code);
```

3.7.1.12 AT91Boot_Read_Data

This function reads X bytes from the connected target.

3.7.1.12.1 Description

*void AT91Boot_Read_Data(int h_handle, int uAddress, char *bValue, int uSize, int *err_code);*

Table 3-11. AT91Boot_Read_Data

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	uAddress	Address where to read 8-bit data
	*bValue	Pointer to an 8-bit data buffer where to store read data
	uSize	Number of bytes to read
Output Parameters	*bValue	Pointer to read data
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF004): USART Communication link not opened • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.12.2 Code Example

```
char bData[10];
AT91Boot_Read_Data(h_handle, 0x200000, bData, 10, &err_code);
```

3.7.1.13 AT91Boot_Go

This function allows starting code execution at specified address.

3.7.1.13.1 Description

*void AT91Boot_Go(int h_handle, int uAddress, int *err_code);*

Table 3-12. AT91Boot_Read_Data

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	uAddress	Address where to start code execution
Output Parameters	none	
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.1.13.2 Code Example

```
AT91Boot_Go(h_handle, 0x200000, &err_code);
```

3.7.2 Internal Flash Programming Functions

These functions are available only for AT91SAM microcontrollers with Flash.

3.7.2.1 AT91Boot_SAM7xxx_Send_Flash

These functions make it possible to write X bytes into the internal Flash memory of the connected target. If some sectors are locked, they are unlocked in order to effectively program the internal Flash memory.

Available functions are:

- AT91Boot_SAM7S32_Send_Flash (available for SAM7S32 and SAM7S321 parts)
- AT91Boot_SAM7S64_Send_Flash
- AT91Boot_SAM7S128_Send_Flash
- AT91Boot_SAM7S256_Send_Flash
- AT91Boot_SAM7S512_Send_Flash
- AT91Boot_SAM7A3_Send_Flash
- AT91Boot_SAM7X128_Send_Flash (available for SAM7X128 and SAM7XC128 parts)
- AT91Boot_SAM7X256_Send_Flash (available for SAM7X256 and SAM7XC256 parts)
- AT91Boot_SAM7X512_Send_Flash (available for SAM7X512 and SAM7XC512 parts)
- AT91Boot_SAM7SE32_Send_Flash
- AT91Boot_SAM7SE256_Send_Flash
- AT91Boot_SAM7SE512_Send_Flash

3.7.2.2 Prerequisite

Embedded Flash Controller Flash Mode Register (EFC_FMR) must be programmed correctly prior to using one of these functions.

Note: Two Embedded Flash Controllers are embedded in AT91SAM7S512, AT91SAM7X512 and AT91SAM7SE512 parts. Both EFC_FMRx registers must be programmed correctly prior to using one of these functions.

3.7.2.2.1 Description

*void AT91Boot_SAM7xxx_Send_Flash(int h_handle, int uOffset, char *bData, int uSize, int *err_code);*

Table 3-13. AT91Boot_SAM7xxx_Send_Flash

Type	Name	Details
Input Parameters	h_handle	Communication handle returned by AT91Boot_Open function
	uOffset	Internal Flash Offset where to write 8-bit value
	*bData	Pointer to 8-bit data buffer to write
	uSize	Buffer size in bytes

Table 3-13. AT91Boot_SAM7xxx_Send_Flash (Continued)

Type	Name	Details
Output Parameters	none	
Error Code	*err_code	• (int)(0x0000) AT91C_BOOT_DLL_OK Standard Error Codes: • (int)(0xF001): Bad h_handle parameter • (int)(0xF002): Address is not correctly aligned • (int)(0xF003): uSize is not correct • (int)(0xF004): USART Communication link not opened • (int)(0xF005): Communication link broken CAN Error Codes: • (int)(0x8002): CAN_Read dll function returned “fail” • (int)(0x8003): CAN_Write dll function returned “fail”
Return Code	void	

3.7.2.2.2 Code Example

```
char bData[10] =
{0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09};

// Write buffer at offset 0x100 into internal SAM7S64 Flash
AT91Boot_SAM7S64_Send_Flash(h_handle, 0x100, bData, 10, &err_code);
```

3.7.3 Using AT91Boot_DLL with MFC

The project OLE_MFC.dsw is located in Examples\OLE_MFC folder. It scans connected devices, opens the first one, reads DBUG chip ID. If an AT91SAM7S256 is detected, it programs a small application (BasicMouse) in the internal Flash.

To use AT91Boot_DLL in such a project, the following steps must be performed:

- Create an AT91Boot_DLL class in your project. To do this, copy both at91boot_dll.cpp and at91boot_dll.h files into your project directory.

  Note: Do not use the ClassWizard/Add Class/From a type library... as there is a bug in Visual C++ 6.0. The bug prevents any functions containing a char variable as a parameter from being imported.

- Initialize OLE libraries by calling `AfxOleInit` function.
- Create an AT91Boot_DLL driver object to manage AT91Boot_DLL COM object.
- Create an AT91Boot_DLL COM object instance with the AT91Boot_DLL program ID[1](`"AT91Boot_DLL.AT91BootDLL.1"`) by using `CreateDispatch` function.

Note: 1. Program ID is stored in the base register and is an easier way to retrieve AT91Boot_DLL Class ID necessary for CreateDispatch function.

Once these four steps have been performed, DLL functions should be available. See their prototypes in at91boot_dll.h header file and for details on how to call these functions.

Note: At this step, if AT91Boot_DLL functions are not available, it is because the AT91Boot_DLL dll has not been registered correctly. See Section 3.4.1 "DLL Registration" on page 3-3 for more information.

3.7.3.1 Code Example

```
#include "at91boot_dll.h"

IAT91BootDLL *m_pAT91BootDLL;

AfxOleInit();

m_pAT91BootDLL = new IAT91BootDLL;

m_pAT91BootDLL->CreateDispatch(_T("AT91Boot_DLL.AT91BootDLL.1"));
```

3.7.4 Using AT91Boot_DLL without MFC

This paragraph explains the project OpenRDI_OLE.dsw located in AT91Boot DLL Example\OLE without MFC folder.

To use AT91Boot_DLL in such a project, the following steps must be performed:

- Initialize COM library by calling `CoInitialize(NULL)`. Use `CoUninitialize()` to close COM Library at the end of the project.
- Import AT91Boot_DLL COM object from AT91Boot_DLL.tlb Type Library file. Eventually rename namespace if necessary.
- Add `using namespace` directive to share the same namespace as AT91Boot_DLL library.
- Create a pointer to AT91Boot_DLL COM objec.t

3.7.4.1 Code Example

In stdafx.h header file

```
#import "YOUR_INSTALL_DIRECTORY/AT91Boot_DLL.tlb" rename_namespace
("AT91BOOTDLL_Lib")
```

In OpenRDI_OLE.cpp source file

```
using namespace AT91BOOTDLL_Lib;

CoInitialize(NULL);

// COM Object Creation
IAT91BootDLLPtr pAT91BootDLL(__uuidof(AT91BootDLL));
```

3.8 AT91Boot_TCL Interface

AT91Boot_TCL.dll must be loaded in order to access its functions. The command is:

```
load [file join AT91Boot_TCL.dll] At91boot_tcl
```

Note: The command is case sensitive.

When SAM-BA starts, a structure containing board and connection information is set. This global variable name is `target`, and the `global target` statement must be declared in any procedure using an API function.

Target structure contents:

- handle: identifier of the link used to communicate with the target
- board: a string containing the board name (i.e.: “AT91SAM7SE512-EK“)
- connection: connection type. Can be: “\usb\ARMx” for a USB link, “COMx” for a serial link (with x indicating the COM port used), or “\jtag\ARMx”
- comType: 1 for serial; 0 for other (USB or JTAG)

These commands are mainly used to send and receive data from the device:

Table 3-14. Commands Available through the TCL Shell

Commands	Argument(s)	Example
TCL_Write_Byte	Handle Value Address err_code	TCL_Write_Byte $target(handle) 0xCA 0x200001*err_code*
TCL_Write_Short	Handle Value Address err_code	TCL_Write_Short $target(handle) 0xCAFE 0x200002 *err_code*
TCL_Write_Int	Handle Value Address err_code	TCL_Write_Int $target(handle) 0xCAFEDECA 0x200000 *err_code*
TCL_Read_Byte	Handle Address err_code	TCL_Read_Byte $target(handle) 0x200003 *err_code*
TCL_Read_Short	Handle Address err_code	TCL_Read_Short $target(handle) 0x200002 *err_code*
TCL_Read_Int	Handle Address err_code	TCL_Read_Int $target(handle) 0x200000 *err_code*
TCL_Go	Handle Address err_code	TCL_Go $target(handle) 0x20008000 *err_code*

These functions are available for all AT91SAM microcontrollers.

- list TCL_Scan
- set h_handle [TCL_Open $name]
- TCL_Close $h_handle
- Write commands: Write a byte (**TCL_Write_Byte**), a half-word (**TCL_Write_Short**) or a word (**TCL_Write_Int**) to the target.
  - *Handle*: handler number of the communication link established with the board.
  - *Value*: byte, half-word or word to write in decimal or hexadecimal.
  - *Address*: address in decimal or hexadecimal.
  - *Output*: nothing.
- Read commands: Read a byte (**TCL_Read_Byte**), a half-word (**TCL_Read_Short**) or a word

(**TCL_Read_Int**) from the target.

- *Handle*: handler number of the communication link established with the board.
- *Address*: address in decimal or hexadecimal.
- *Output*: the byte, half-word or word read in decimal.

Note: **TCL_Read_Int** returns a signed integer in decimal. For example, reading with **TCL_Read_Int** $target(handle) *0xFFFFFFFF* command returns *-1* whereas reading *0xFF* with **TCL_Read_Byte** command returns *255*.

- **send_file**: Send a file to a specified memory.
  - *Memory*: memory tag (in curly brackets).
  - *fileName*: absolute path file name (in quotes) or relative path from the current directory (in quotes).
  - *Address*: address in decimal or hexadecimal.
  - *Output*: information about the corresponding command on the TCL Shell.
- **receive_file**: Receive data into a file from a specified memory.
  - *Memory*: memory tag (in curly brackets).
  - *fileName*: absolute path file name (in quotes) or relative path (from the current directory) file name (in quotes).
  - *Address*: address in decimal or hexadecimal.
  - *Size*: size in decimal or hexadecimal.
  - *Output*: information about the corresponding command on the TCL Shell.
- **compare_file**: Compare a file with memory data.
  - *Memory*: memory tag (in curly brackets).
  - *fileName*: absolute path file name (in quotes) or relative path from the current directory (in quotes).
  - *Address*: address of the first data to compare with the file in decimal or hexadecimal.
  - *Output*: information about the command progress on the TCL Shell.
- **TCL_Compare**: Binary comparison of two files.
  - *fileName1*: absolute path file name (in quotes) or relative path from the current directory (in quotes) of the first file to compare.
  - *fileName2*: absolute path file name (in quotes) or relative path from the current directory (in quotes) of the second file to compare with the first.
  - *Output*: return 1 in case of error, 0 if files are identical.
- **TCL_Go**: Jump to a specified address and execute the code.
  - *Handle*: handler number of the communication link established with the board.
  - *Address*: address to jump to in decimal or hexadecimal.

The memory tag is the name (in curly brackets) of the memory module defined in the memoryAlgo array in the board description file, e.g.: `{DataFlash AT45DB/DCB}`

Moreover, a set of older commands (for SAM-BA v1.x script compatibility) is always available:

Table 3-15. Extended Command Set

Commands	Argument(s)	Example
write_byte	*Address Value*	**write_byte** 0x200001 0xCA
write_short	*Address Value*	**write_short** 0x200002 0xCAFE
write_int	*Address Value*	**write_int** 0x200000 0xCAFEDECA
read_byte	*Address*	**read_byte** 0x200003
read_short	*Address*	**read_short** 0x200002
read_int	*Address*	**read_int** 0x200000
go	*Address*	**go** 0x20008000

- Write commands: Write a byte (**write_byte**), a half-word (**write_short**) or a word (**write_int**) to the target.
  - *Address*: address in decimal or hexadecimal.
  - *Value*: byte, half-word or word to write in decimal or hexadecimal.
  - *Output*: nothing.
- Read commands: Read a byte (**read_byte**), a half-word (**read_short**) or a word (**read_int**) from the target.
  - *Address*: address in decimal or hexadecimal.
  - *Output*: the byte, half-word or word read in decimal.

  Note: **read_int** returns a signed integer in decimal. For example, reading with **read_int** *0xFFFFFFFF* command returns *-1* whereas reading *0xFF* with **read_byte** command returns *255*.
- **go**: Jump to a specified address and execute the code.
  - *Address*: address to jump to in decimal or hexadecimal.

Warning: The send_file, receive_file and compare_file commands are slightly different now. The Memory tag syntax changed (see Table 3-15).

If you leave SAM-BA, be sure to reboot your board before launching it the next time.

3.9 SAM-BA TCL Interface

Table 0-1. Commands Available through the TCL Shell

Commands	Argument(s)	Example
send_file	{Memory} “fileName” Address	send_file {SDRAM} “C:/temp/file1.bin“ 0x20000000
receive_file	{Memory} “fileName” Address Size	receive_file {SDRAM} “C:/temp/file1.bin“ 0x10000256
compare_file	{Memory} “fileName” Address	compare_file {SDRAM} “C:/temp/file1.bin“ 0x20000000

Section 4

SAM-BA Customization

4.1 Overview

SAM-BA can be easily customized to add a new AT91SAM based board or to add a new programming algorithm for a new device. All AT91SAM evaluation kit descriptions are freely available in the AT91ISP installation directory. Likewise, all programming algorithms sources are available under the AT91ISP installation directory.

4.2 Adding a New Board

SAM-BA customization allows the user to add new window tabs dedicated to particular memories in the GUI. Figure 4-3 shows a board with 5 memory modules described.

4.2.1 Adding a Board Entry

Customization of SAM-BA GUI for a particular board/device is done by adding new TCL scripts in the directory : [*Install Directory*]\lib.

A good starting point is to have a look at the TCL files provided with SAM-BA.

The file `boards.tcl` contains the board names and associated description file paths.

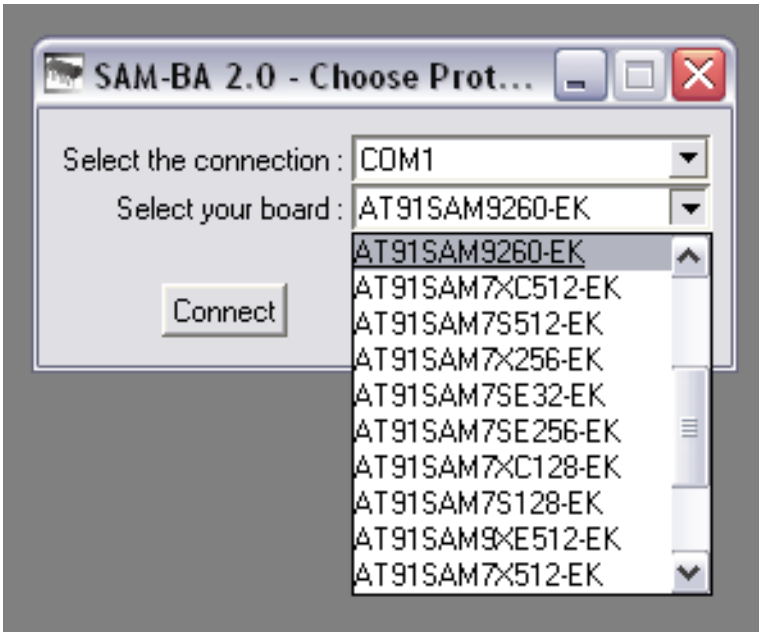
To add support for a new board, a new entry must be created in the `boards` array and the corresponding directory. The directory must have the same name as the board.

```
array set boards {
"NO_BOARD""NO_BOARD/NO_BOARD.tcl"
"AT91SAM7A3-EK"   "AT91SAM7A3-EK/AT91SAM7A3-EK.tcl"
"AT91SAM7S32-EK"  "AT91SAM7S32-EK/AT91SAM7S32-EK.tcl"
"AT91SAM7S64-EK"  "AT91SAM7S64-EK/AT91SAM7S64-EK.tcl"
"AT91SAM7S128-EK" "AT91SAM7S128-EK/AT91SAM7S128-EK.tcl"
"AT91SAM7S256-EK" "AT91SAM7S256-EK/AT91SAM7S256-EK.tcl"
"AT91SAM7S512-EK" "AT91SAM7S512-EK/AT91SAM7S512-EK.tcl"
"AT91SAM7S321-EK" "AT91SAM7S321-EK/AT91SAM7S321-EK.tcl"
"AT91SAM7SE32-EK" "AT91SAM7SE32-EK/AT91SAM7SE32-EK.tcl"
"AT91SAM7SE256-EK""AT91SAM7SE256-EK/AT91SAM7SE256-EK.tcl"
"AT91SAM7SE512-EK""AT91SAM7SE512-EK/AT91SAM7SE512-EK.tcl"
"AT91SAM7X128-EK" "AT91SAM7X128-EK/AT91SAM7X128-EK.tcl"
"AT91SAM7X256-EK" "AT91SAM7X256-EK/AT91SAM7X256-EK.tcl"
"AT91SAM7X512-EK" "AT91SAM7X512-EK/AT91SAM7X512-EK.tcl"
"AT91SAM7XC128-EK""AT91SAM7XC128-EK/AT91SAM7XC128-EK.tcl"
```

```
"AT91SAM7XC256-EK""AT91SAM7XC256-EK/AT91SAM7XC256-EK.tcl"
"AT91SAM7XC512-EK""AT91SAM7XC512-EK/AT91SAM7XC512-EK.tcl"
"AT91SAM9260-EK"  "AT91SAM9260-EK/AT91SAM9260-EK.tcl"
"AT91SAM9261-EK"  "AT91SAM9261-EK/AT91SAM9261-EK.tcl"
"AT91SAM9263-EK"  "AT91SAM9263-EK/AT91SAM9263-EK.tcl"
"AT91SAM9XE512-EK""AT91SAM9XE512-EK/AT91SAM9XE512-EK.tcl"
"NEW_BOARD"       "NEW_BOARD/NEW_BOARD.tcl"
}
```

The first field is the board name (the name that appears in the “Select your board” listbox when SAM-BA is started), and the second is the directory where the board and the memory module description files are located.

Figure 4-1. Adding a New Board



Each subdirectory corresponds to one board, and has **one board description file** (`[board_name].tcl`), and applet binaries(i.e., `isp-xx-[device_name].bin`).

For example, the directory of the AT91SAM7SE512-EK board contents are:

[*Install Directory*]`\SAM-BA v2.x\lib /AT91SAM7SE512-EK:`

```
AT91SAM7SE512-EK.tcl
isp-dataflash-at91sam7se512.bin
isp-extram-at91sam7se512.bin
isp-flash-at91sam7se512.bin
isp-serialflash-at91sam7se512.bin
isp-nandflash-at91sam7se512.bin
isp-norflash-at91sam7se512.bin
```

Source code of each applet is located in the [*Install Directory*]\SAM-BA v2.7\applets directory.

4.2.2 Board Description File

Board description files accomplish the link between applets and generic transfer routines running on the host PC.

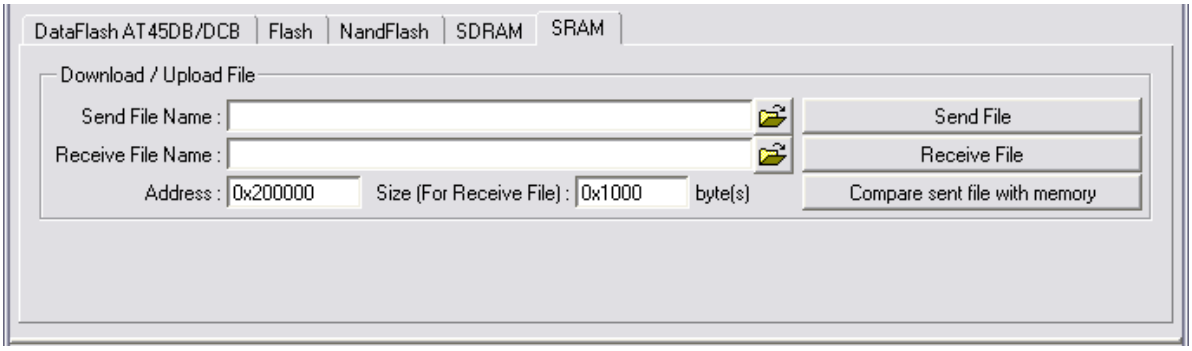
Several hash tables list memory algorithms which apply to the board and parameters.

The first array found in the board description file lists the memory modules present on the board. Note that some devices such as Peripheral or REMAP can also be found here, but are just address ranges displayed in the Memory Display window.

```
array set memoryAlgo {
    "SRAM"                    "::at91sam7se512_sram"
    "SDRAM"                   "::at91sam7se512_sdram"
    "Flash"                   "::at91sam7se512_flash"
    "DataFlash AT45DB/DCB"    "::at91sam7se512_dataflash"
    "NandFlash"               "::at91sam7se512_nandflash"
    "NorFlash Map"            "::at91sam7se512_norflash_map"
    "Peripheral"              "::at91sam7se512_peripheral"
    "ROM"                     "::at91sam7se512_rom"
    "REMAP"                   "::at91sam7se512_remap"
}
```

The first 5 entries correspond to the five Memory Download tabs (see Figure 4-3).

Figure 4-2. Memory Download Tabs



A memory module array is defined for each module declared in the memoryAlgo array:

```
array set at91sam7se512_nandflash {
  dftDisplay  1
  dftDefault 1
  dftAddress  0x0
  dftSize     0x10000000
  dftSend     "NANDFLASH::sendFile"
  dftReceive  "NANDFLASH::receiveFile"
  dftScripts  "::at91sam7se512_nandflash_scripts"
}
```

Field definitions:

- dftDisplay: indicates if the memory appears as a Memory Download tab (0: no, 1: yes).
- dftDefault: Set to one if this memory tab shall be selected when SAM-BA starts. (There shall be one default memory tab among all memory tabs).
- dftAddress: base address of the memory module. (0x0 for memories not physically mapped, like DataFlash, or when accesses are not directly done, but need a monitor, like NAND Flash).
- dftSize: size of the memory module.
- dftSend: send file procedure name.
- dftReceive: receive file procedure name.
- dftScripts: name of the array containing the script list, see Figure 4-3 below, (blank if no script is implemented).

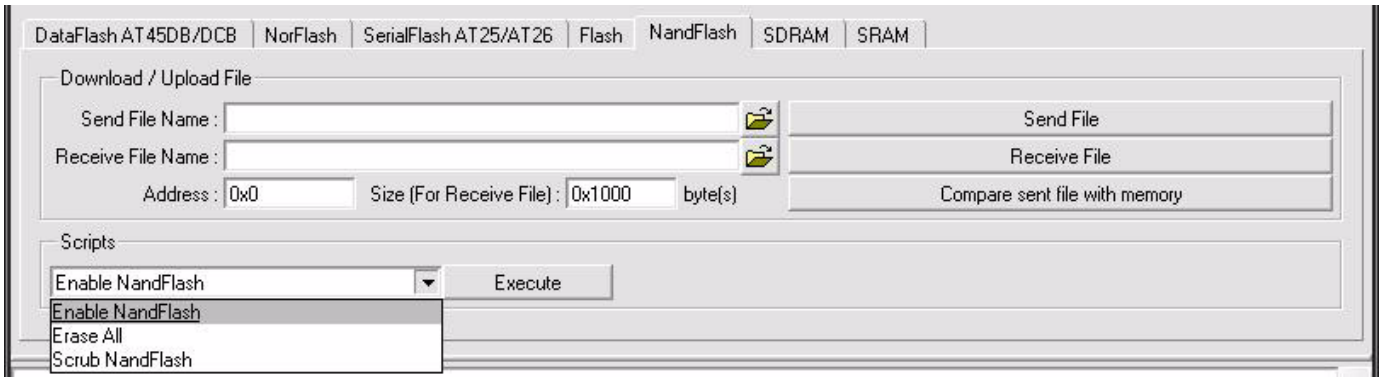
Scripts can be implemented for each memory module. Common uses are SDRAM initialization, Flash erase operation, or any other frequently used operation. The scripts are displayed in the script listbox of the corresponding memory tab. Their declaration is done by creating an array named in the dftScripts field of the memory:

```
array set at91sam7se512_nandflash {
    dftDisplay  1
    dftDefault  0
    dftAddress  0x0
    dftSize     "$GENERIC::memorySize"
    dftSend     "GENERIC::SendFile"
    dftReceive  "GENERIC::ReceiveFile"
    dftScripts  "::at91sam7se512_nandflash_scripts"
}

array set at91sam7se512_nandflash_scripts {
    "Enable NandFlash"    "NANDFLASH::Init"
    "Erase All"           "GENERIC::EraseAll"
    "Scrub NandFlash"     "GENERIC::EraseAll $NANDFLASH::scrubErase"
}
set NANDFLASH::appletAddr       0x20000000
set NANDFLASH::appletFileName   "$libPath(extLib)/$target(board)/isp-
nandflash-at91sam7se512.bin"
```

The first field of each entry is the string displayed in the listbox, and the second is the procedure name invoked when executing the script.

Figure 4-3. Memory Scripts



4.3 Extending SAM-BA Programming Capabilities

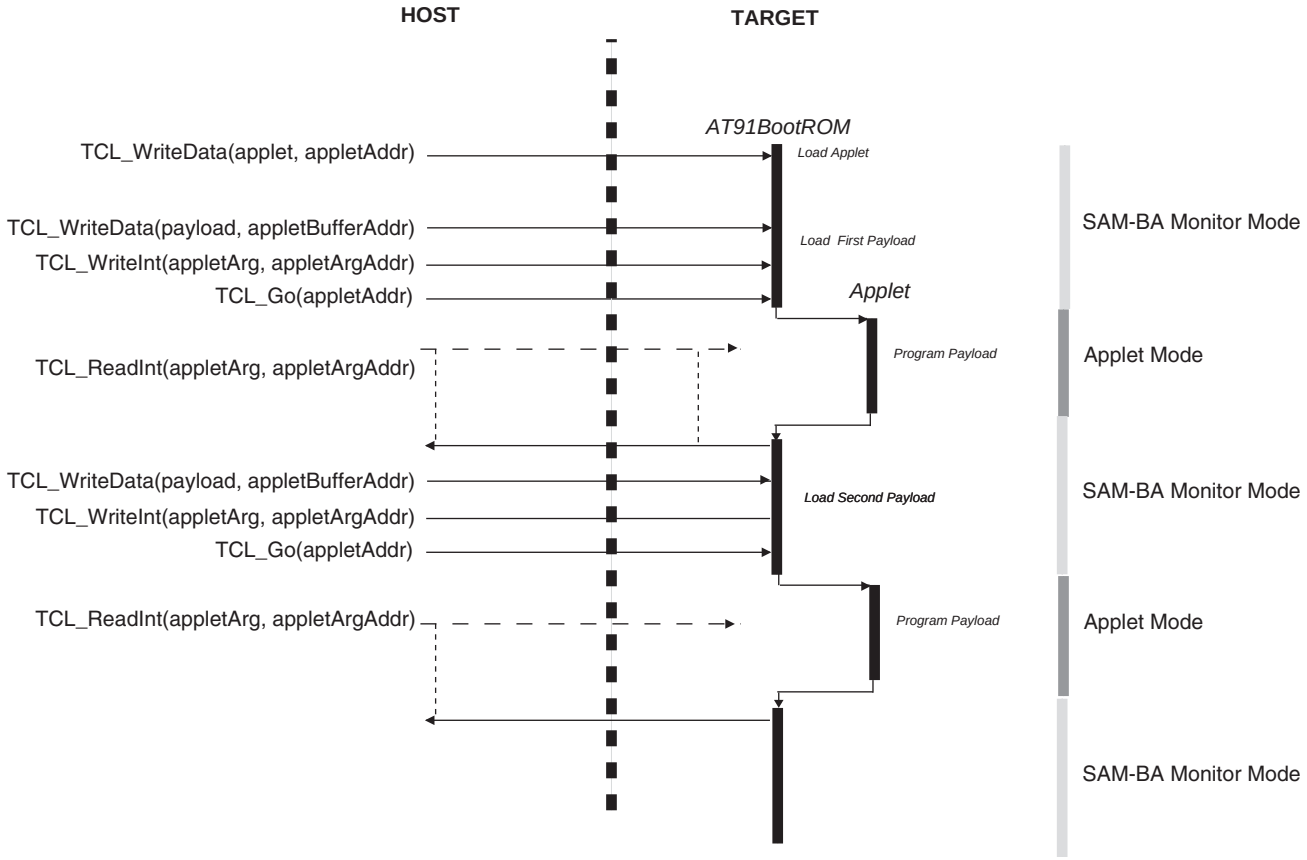
Several programming algorithms (applets) are delivered in the standard AT91ISP installation. They are pieces of C code, that are compiled specifically for each chip. However, these programming algorithms may require some adaptation to match a new board specification. This is the case in particular when a memory is not connected to the same PIO or the same chip select as on the evaluation kit.

4.3.1 Applet Workflow

The target handles the programming algorithm by running applets. The target switches between two modes: **SAM-BA Monitor Mode** and **Applet Mode**. The SAM-BA monitor mode is the command interpreter that runs in the ROM memory when you connect the chip with USB or COM port to the computer. It allows the computer to send or receive data to/from the target. All transfers between host and device are done when the device is in SAM-BA monitor mode. Under Applet Mode, the device performs programming operations and is not able to communicate with the host.

An applet is a small piece of software running on the target. It is loaded in the device memory while the device is in SAM-BA monitor mode using TCL_Write command. The device switches from SAM-BA monitor mode to Applet mode using the TCL_Go command. The device executes the applet code. At the end of the current operation, the device switches back to SAM-BA monitor mode.

Figure 4-4. Applet Workflow



An applet can execute different programming or initialization commands. Before switching to Applet mode, the host prepares command and arguments data required by the applet in a mailbox mapped in the device memory. During its execution, the applet decodes the commands and arguments prepared by the host and executes the corresponding function. The applet returns state, status and result values in the mailbox area.

Usually, applets include INIT, buffer read, buffer write functions. To program large files, the whole programming operation is split by the host into payloads. Each payload is sent to a device memory buffer using SAM-BA monitor command TCL_Write. The host prepares the mailbox with the Buffer write command value, the buffer address and the buffer size. The host then forces the device in Applet mode using a TCL_Go command. The host polls the end of payload programming by trying to read the state value in the mailbox. The device will answer to the host as soon as it returns to SAM-BA monitor mode.

In case of USB connection, when the host polls while the device is in Applet mode, the device NACK IN packets sent by the host. Applet execution has to be short enough in order to prevent from connection timeout error. In case of long programming or erasing operation, from time to time, the device shall leave Applet mode to return to SAM-BA monitor mode in order to be able to achieve the current pending host TCL_ReadInt command within the timeout threshold.

4.3.2 SDRAM Initialization

To extend the buffer size, some applets run in SDRAM. This requires SDRAM initialization. SDRAM initialization is done by an applet. C code of this applet is located in the "[*Install Directory*]\SAM-BA

v2.x\applets\isp-applets\extram" directory. In case the applet fails to initialize it, SAM-BA launch will stop with the following error message.

Figure 4-5. Error Message

Usually, SDRAM initialization is done by default during SAM-BA start-up sequence in the <board>.tcl file.

This applet may require customization to support a new board SDRAM configuration. The initialization is done in the <board>.tcl file:

```
# Initialize SDRAMC
if {[catch {GENERIC::Init $SDRAM::appletAddr $RAM::appletFileName {}}
dummy_err] } {
        puts "-E- Error during SDRAM initialization"
} else {
        puts "-I- SDRAM initialized"
}
```

If a custom board has different SDRAM characteristics (timing, pinout, ...) it is mandatory to recompile the “extram” applet with correct settings.

4.3.3 Applets Usage

Each memory programming interface is described in an array in <board>.tcl. Please refer to Section 4.2.2 ”Board Description File” on page 4-3.

Applets are usually loaded by an INIT script. This TCL procedure belongs to the memory namespace, DATAFLASH to set an example. DATAFLASH::Init invokes GENERIC::Init procedure with the applet binary location, the loading address and eventually arguments required by the INIT function of the data-flash applet, SPI index and chip select index to set an example. The INIT function returns the payload buffer address. This payload buffer address stored in GENERIC::appletBufferAddress is used in the GENERIC::Write function to send data to be programmed to the device. The INITfunction also returns the memory size and the buffer payload size: GENERIC::appletBufferSize.

TCL procedure GENERIC::Init loads applet binary in the target calling TCL_Write_Data to the applet link address (appletAddr). Once the applet is loaded, a TCL_Go forces the target to leave the SAM-BA monitor and start applet execution. At the end of the applet operations, the target resumes the SAM-BA monitor execution.

DATAFLASH Applet initialization example:

```
################################################################################
## DATAFLASH
################################################################################

array set at91sam7se512_dataflash {
    dftDisplay  1
```

```
    dftDefault  1
    dftAddress  0x0
    dftSize     "$GENERIC::memorySize"
    dftSend     "GENERIC::SendFile"
    dftReceive  "GENERIC::ReceiveFile"
    dftScripts  "::at91sam7se512_dataflash_scripts"
}

array set at91sam7se512_dataflash_scripts {
    "Enable Dataflash (SPI0 CS0)"                          "DATAFLASH::Init 0"
    "Set DF in Power-Of-2 Page Size mode (Binary mode)"  "DATAFLASH::BinaryPage"
    "Erase All"                                            "DATAFLASH::EraseAll"
}

set DATAFLASH::appletAddr       0x20000000
set DATAFLASH::appletFileName   "$libPath(extLib)/$target(board)/isp-dataflash-at91sam7se512.bin"




#=============================================================================
#  proc DATAFLASH::Init
#-----------------------------------------------------------------------------
proc DATAFLASH::Init {dfId} {
    global target
    variable appletAddr
    variable appletFileName
    variable DATAFLASH_initialized

    set DATAFLASH_initialized 0

    puts "-I- DATAFLASH::Init $dfId (trace level : $GENERIC::traceLevel)"

    # Load the applet to the target
    if {[catch {GENERIC::Init $DATAFLASH::appletAddr $DATAFLASH::appletFileName [list
$target(comType) $GENERIC::traceLevel $dfId ]} dummy_err] } {
        error "Error Initializing DataFlash Applet ($dummy_err)"
    }

    set DATAFLASH_initialized 1
}




#=============================================================================
#  proc GENERIC::Init
```

```
#--------------------------------------------------------------------------------
proc GENERIC::Init {memAppletAddr appletFileName {appletArgList 0}} {
    global target
    variable appletAddr
    variable appletCmd

    # Update the current applet address
    set appletAddr $memAppletAddr

    # Load the applet to the target
    if {[catch {GENERIC::LoadApplet $appletAddr $appletFileName} dummy_err]} {
        error "Applet $appletFileName can not be loaded"
    }

    # Run the INIT command
    set appletAddrCmd       [expr $appletAddr + 0x04]
    set appletAddrArgv0     [expr $appletAddr + 0x0c]
    set appletAddrArgv1     [expr $appletAddr + 0x10]
    set appletAddrArgv2     [expr $appletAddr + 0x14]
    set appletAddrArgv3     [expr $appletAddr + 0x18]


    # Write the Cmd op code in the argument area
    if {[catch {TCL_Write_Int $target(handle) $appletCmd(init) $appletAddrCmd} dummy_err] } {
        error "Error Writing Applet command\n$dummy_err"
    }

    set argIdx 0
    foreach arg $appletArgList {
        # Write the Cmd op code in the argument area
        if {[catch {TCL_Write_Int $target(handle) $arg [expr $appletAddrArgv0 + $argIdx]}
dummy_err] } {
            error "Error Writing Applet argument $arg ($dummy_err)"
        }
        incr argIdx 4
    }

    # Launch the applet Jumping to the appletAddr
    if {[catch {set result [GENERIC::Run $appletCmd(init)]} dummy_err]} {
        error "Applet Init command has not been launched ($dummy_err)"
    }
    if {$result == 1} {
        error "Can't detect known device"
    } elseif {$result != 0} {
        error "Applet Init command returns error: [format "0x%08x" $result]"
    }

    # Retrieve values
```

```
    variable memorySize
    variable appletBufferAddress
    variable appletBufferSize
    set GENERIC::memorySize          [TCL_Read_Int $target(handle) $appletAddrArgv0]
    set GENERIC::appletBufferAddress [TCL_Read_Int $target(handle) $appletAddrArgv1]
    set GENERIC::appletBufferSize    [TCL_Read_Int $target(handle) $appletAddrArgv2]


    set FLASH::flashLockRegionSize [expr [TCL_Read_Int $target(handle) $appletAddrArgv3] &
0xFFFF]
    set FLASH::flashNumbersLockBits  [expr [TCL_Read_Int $target(handle) $appletAddrArgv3] >> 16]
    set FLASH::flashSize $GENERIC::memorySize

    puts "-I- Memory Size : [format "0x%X" $GENERIC::memorySize] bytes"
    puts "-I- Buffer address : [format "0x%X" $GENERIC::appletBufferAddress]"
    puts "-I- Buffer size: [format "0x%X" $GENERIC::appletBufferSize] bytes"

    puts "-I- Applet initialization done"
}



#================================================================================
#  proc GENERIC::LoadApplet
#--------------------------------------------------------------------------------
proc GENERIC::LoadApplet {appletAddr appletFileName} {
    global target
    global libPath
    if {$target(connection) != {\usb\ARM0} && $target(connection) != {\jtag\ARM0}} {
        set GENERIC::traceLevel 5
    }
    puts "-I- Loading applet [file tail $appletFileName] at address [format "0x%X" $appletAddr]"

    # Open Data Flash Write file
    if { [catch {set f [open $appletFileName r]}] } {
        error "Can't open file $appletFileName"
    }

    # Copy applet into Memory at the  appletAddr
    fconfigure $f -translation binary
    set size [file size $appletFileName]
    set appletBinary [read $f $size]
    if {[catch {TCL_Write_Data $target(handle) $appletAddr appletBinary $size dummy_err}
dummy_err]} {
        error "Can't write applet $appletFileName"
    }
    close $f
}
```

```
#=============================================================================
#  proc GENERIC::Run
#
#  Launch the applet, wait for the end of execution and return the result
#-----------------------------------------------------------------------------
proc GENERIC::Run { cmd } {
    global target
    variable appletAddr

    set appletCmdAddr    [expr $appletAddr + 4]
    set appletStatusAddr [expr $appletAddr + 8]

    puts "-I- Running applet command $cmd at address [format "0x%X" $appletAddr]"

    # Launch the applet Jumping to the appletAddr
    if {[catch {TCL_Go $target(handle) $appletAddr} dummy_err] } {
        error "Error Running the applet"
    }

    # Wait for the end of execution
    # TO DO: Handle timeout error
    set result $cmd
    while {$result != [expr ~($cmd)]} {
        if {[catch {set result [TCL_Read_Int $target(handle) $appletCmdAddr]} dummy_err] } {
            error "Error polling the end of applet execution"
        }
    }

    # Return the error code returned by the applet
    if {[catch {set result [TCL_Read_Int $target(handle) $appletStatusAddr]} dummy_err] } {
        error "Error reading the applet result"
    }

    return $result
}
```

4.3.4 TCL Wrappers for Read and Write Applets

TCL procedures GENERIC::SendFile and GENERIC::ReceiveFile invoke the GENERIC::Write and GENERIC::Read TCL procedures. These procedures split data into payloads of GENERIC::appletBufferSize size.

TCL procedure GENERIC::Write loads data payload in the target calling TCL_Write_Data to the GENERIC::appletBufferAddress. The WRITE command is set in the first mailbox word by the TCL_Write_Int TCL command. TCL_Go forces the target to leave the SAM-BA monitor and start applet WRITE execution. At the end of the applet operations, the target resumes the SAM-BA monitor execu-

tion. The host polls the end of the programming operation, reading the first word of the mailbox. Once the device has answered, the host can send the rest of the data by sending the next payload.

4.3.5 Applet Compilation

An applet is running on the target device. Thus an applet consists in ARM binary code. It is loaded by a monitor ”WriteData command” and launched by a monitor “go command”.

All applet source code is delivered in the SAM-BA installation directory:
[*Install Directory*]\SAM-BA v2.x\applets\isp-applets

Applets can be compiled using GNU tools. Each applet is delivered with a standard Makefile. This Makefile takes in arguments of board and chip names. A perl script file:[*Install Directory*]\SAM-BA v2.x\applets\isp-applets\build.pl provides all commands executed to compile all applet configurations. Thus the same algorithm is used on different devices.

Resulting binary shall be copied into the corresponding board library: [*Install Directory*]\SAM-BA v2.x\lib.

To get an example, please refer to the customization example provided with your specific SAM-BA version.

4.3.6 Applet Internal Structure

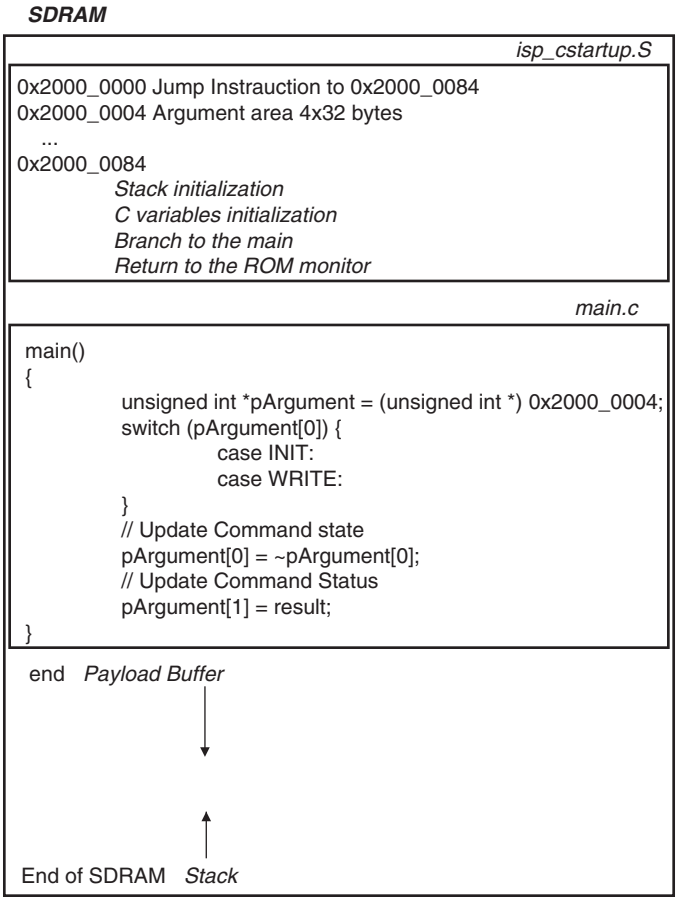
During startup sequence, the BSS segment shall be reset to 0. Once loaded, the applet may be invoked several times to execute the same or different functions. To keep global variable values, the BSS segment initialization must be done only once. In the isp_cstartup.S file, its Initialized variable is tested to prevent multiple BSS initialization.

The stack pointer is initialized in the boot ROM. However, on some AT91SAM device revisions, each time the SAM-BA Monitor mode is left by a go command, the stack pointer is not reset to its initial value which produces a memory leakage. The workaround is to initialize the stack pointer each time the applet is entered. At the end of the applet, boot ROM is resumed with the stack pointer set to the applet’s stack pointer initial value.

Each time the boot ROM executes a go command, it resets PIO initialization. Applets must take care of that and perform PIO initialization each time it is resumed.

A mailbox shared between the applet and host application is located at the beginning of the execution region, just after the jump instruction.Then it is easy for the host application to determine where the mailbox is located: applet load address + 4 bytes.

Figure 4-6. Applet Mapping



The 32 4-byte words mailbox definition must be shared between the applet and the host application. By default, the first word of the mailbox initialized by the host application corresponds to the applet function ID. The first word of the mailbox set by the applet corresponds to the logical inversion of the function ID. The other words may be used as applet function arguments. The first word is used by the application to determine that the applet function is achieved. The second word of the mailbox set by the applet corresponds to the result of the applet function. Other words can be used as values returned by the applet function.

Memory space located after the applet binary code can be used as a dedicated area to store buffer payloads received in Boot ROM mode and programmed in the media by the applet. A good practice is to implement an applet INIT function which returns the address of this memory space.

Section 5

Revision History

5.1 Revision History Table

Table 5-1.

Document Reference	Comments	Change Request Ref.
6421A	This document is intended to replace previously published user guides and application note: Atmel lit° 6132, 6224, 6272	
	First Issue	

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com
www.atmel.com/AT91SAM

Technical Support
AT91SAM Support
Atmel techincal support

Sales Contacts
www.atmel.com/contacts/

Literature Requests
www.atmel.com/literature